ARMY RESEARCH LABORATORY

Establishing an Experimental Testbed with Software-defined Radios

by Gunjan Verma and Paul Yu

ARL-TR-5896 **January 2012**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Adelphi, MD 20783-1197

ARL-TR-5896 **January 2012**

Establishing an Experimental Testbed with Software-defined Radios

Gunjan Verma and Paul Yu
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
January 2012	Final	

4. TITLE AND SUBTITLE
Establishing an Experimental Testbed with Software-defined Radios

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Gunjan Verma and Paul Yu

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-CIN-T
2800 Powder Mill Road
Adelphi, MD 20783-1197

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-5896

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Software-defined radios (SDRs) provide researchers with a powerful and flexible wireless communications experimentation platform. GNU Radio is the most popular open-source software toolkit for deploying SDRs, and is frequently used with the Universal Software Radio Peripheral (USRP). There is significant complexity involved in setting up GNU Radio with the USRP; therefore, in this report, we describe the key steps to take (and pitfalls to avoid) in order to successfully establish such an SDR testbed.

15. SUBJECT TERMS
GNU Radio, USRP, software defined radio, testbed

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Gunjan Verma
Unclassified	Unclassified	Unclassified	UU	20	19b. TELEPHONE NUMBER (*Include area code*) (301) 394-3102

Contents

List of Tables

1. Introduction

There are two "big ideas" behind the software-defined radio (SDR), which make it so attractive for research in wireless communications:

1. Reconfigurability: With a single radio frontend, SDR uses software packages to enable multiple capabilities previously available only through separate radios.
2. Development time: Only high-level software adjustments are required to prototype and test functionality. This is in comparison to the previous need for circuit design and device-specific code.

Taken together, these two advantages can be summarized as SDR turns highly specialized, device-specific hardware problems into general, device-independent software problems.

In this report, we provide suggestions to experimenters for setting up a GNU Radio SDR platform. GNU Radio is an open-source software toolkit started in 2001 for developing SDR systems (*1*). We focus on the subtleties of installation and configuration that are undocumented, or otherwise difficult to find, in the available documentation.

GNU Radio provides implementations of many signal processing algorithms (called GNU Radio **blocks**) in C++, with interfaces to some of these blocks available in Python. Building a custom radio is as easy as connecting the signal processing blocks (e.g., modulator to mixer to amplifier) together in Python. If a signal processing capability is not available, a new block can be implemented in C++.

Although in principle, GNU Radio can be used with almost any hardware digital acquisition system (DAQ), the best-supported system is the Universal Software Radio Peripheral (USRP) made by National Instruments (*2*). In this report, we address the use of the USRP (first generation) with GNU Radio. The first-generation USRP is a DAQ with the following features:

- Four 64-megasample per second, 12-bit analog-to-digital converters
- Four 128-megasample per second, 14-bit digital-to-analog converters
- USB 2.0 interface
- Up to 16 MHz wide bandwidth

Various plug-in daughterboards are available to cover spectrum between 0 and 5.9 GHz.

2. Preliminaries

2.1 Software Overview

The GNU Radio package can be obtained from the official Web site (*1*). Before we delve into configuration specifics, it is useful to take a bird's eye view of the GNU Radio software infrastructure. Table 1 lists the main purpose of each key package within the root folder.

Table 1. Key packages of GNU radio.

GNU Radio modules	Description
gr-audio-*	Soundcard support (acts as DAC and signal sink)
gnuradio-core	Code of the core GNU Radio library
gnuradio-examples	Example applications (C++ and Python)
gr-utils	Useful applications like signal generators, IQ plots
gr-usrp	Glue clode connecting USRP to GNU rdaio framework
gr-wxgui	wxPython-based GUI tools (including graphical FFT, oscilloscope)

It is also useful to familiarize oneself with some of the key packages that GNU Radio depends on. While most users will not use these packages directly, they provide key functionality behind the scenes to the GNU Radio software system. Table 2 presents an overview of the most important dependencies of GNU Radio.

Table 2. Key package dependencies of GNU radio.

Package name	Description
cppunit	Unit testing framework for C++
Boost	Software libraries extending functionality of C++
FFTW	High performance C subroutine that computes the DFT
GSL	Software library for numerical computation
Numpy	Python library that adds high performance arrays, linear algebra, and Fourier transform algorithms
sdcc	C-compiler used to build firmware for USRP
SWIG	Tool that exposes code written in C/C++ to Python
wxPython	GUI toolkit for Python

2.2 Which Version to Install

GNU Radio is a rapidly evolving project. While it may seem that choosing the latest version is a surefire bet, some words of caution are in order. For example, versions of GNU

Radio 3.2.2 and earlier used *OMNIthread* to provide threading functionality. *OMNIthread* is a C++ library providing a thread abstraction, namely, a set of thread operations for use in C++ programs. GNU Radio 3.3.0, the latest version, abandons *OMNIthread* in favor of *Boost* threads. Therefore, any application using threading that was written for GNU Radio 3.2.2 or earlier **will not work** with GNU Radio 3.3.0. Other changes that are always being made are the re-implementation of Python blocks into C++; in fact, Python's role in GNU Radio will be completely obsolete in the near future (*3*).

2.3 Use of Virtual Machines

If one plans on designing one's own signal processing blocks and/or modifying existing blocks, we have found that installing GNU Radio inside a virtual machine (VM) (such as provided by virtualization software like KVM or VMWare) is a very good option. Each VM can edit the GNU Radio source code base, for a given application, independent of other VMs. Deployment of the same image over multiple machines is trivial with VMs.

2.3.1 Single Machine with Multiple USRPs

The VM approach permits an easy way of using multiple USRPs on a single machine, where each USRP is associated with a separate VM running its own GNU Radio application. Note that the machine must have sufficiently high processing capability and memory available, otherwise significant slowdown in the form of dropped packets will occur.

When multiple USRPs are installed on a single host, the way to distinguish between them in Python code is as follows:

```
usrp1 = usrp.source_c(which=0)
usrp2 = usrp.source_c(which=1)
```

However, which USRP gets which code (0 or 1) depends on how they appear on the universal serial bus (USB) and device, which is affected by which USRP is plugged in or turned on before the other. This introduces some non-determinism in the numbering of multiple USRPs on a single host machine. One workaround is to query for the serial number of the USRP one is working with, and then take an action accordingly. However, this is an awkward solution.

Virtualization makes it easy to manage to multiple USRPs connected to a single machine; simply assign each USRP to its own VM using the VM software to selectively connect a single USRP (which is viewed by the VM software as just a USB device) to a particular VM instance. This makes testing of new signal processing blocks requiring multiple USRPs

extremely easy, so long as one realizes that the 32 MB/s USB bandwidth is being shared across all the USRPs connected to the host. However, in our experience, in applications with modest bandwidth requirements involving multiple USRPs, the most rapid development environment for situations involving the use of multiple USRPs is as follows:

1. Connect the USRPs to a single host.
2. Assign each USRP to its own VM.
3. Configure each VM to mount a directory from the host operating system (OS) (e.g., call it "common_files").
4. Place any code to be shared across all USRPs in "common_files".

Now, all VMs have access to the common code (e.g., a new signal processing block) in a single step, and each can be launched with the appropriate code from a single directory. This circumvents the need to maintain code synchronization across multiple machines, each connected to a single USRP.

2.3.2 Single Machine with Multiple VMs

The VM approach also allows side-by-side installation of GNU Radio 3.2.2 and 3.3.0, respectively, in two separate VMs. In a setup in which there is a mix of USRPs, USRP2s, and/or other radio devices, one VM containing an installation of GNU Radio can be created and customized for each radio device. There are several other advantages of using VMs, including the ability to take snapshots of known functional system states and roll back changes as needed. This is particularly relevant because software development in GNU Radio, unlike in conventional software development systems, does not allow for interactive debug sessions (since C++ libraries must be built for use from Python), so the likelihood of breaking the GNU radio codebase during development or modification of signal processing algorithms is essentially guaranteed.

3. GNU Radio Installation and USRP Configuration

Once one has decided which version of GNU Radio to install and whether or not to use virtual machines, one is ready to begin.

Basic GNU Radio installation guides are available from the official Web site (*1*). However, we supplement this with additional useful information not covered in the official

documentation. Some distributions, like Ubuntu, offer the GNU Radio package for installation through a package manager, which circumvents the need for manual installation. However, there are several advantages to manual installation, which include the following:

- Package managers only list stable releases (as determined by the repository curator), so newer (or older) versions of GNU Radio may not be available through them.
- For testing purposes, we may desire to have multiple versions of GNU Radio installed side by side.
- Source-code modifications of the underlying signal processing blocks (for example, to test out new algorithms) require manual re-builds anyway, so familiarity with this process is essential.

In each of the following sections, we present the exact installation procedure for Ubuntu 10.04.

3.1 Prerequisites

The complete list of prerequisites can be found from the install guide (*4*). In our experience, we have found GNU Radio to be fairly delicate with respect to the versions of the various software modules installed. For example, compilation errors may occur when using newer versions of the g++ compiler. In addition, the exact prerequisites needed can vary considerably between different distributions of Linux. It is therefore highly recommended to consult the GNU Radio Web site for distributions with explicit configuration instructions.

For Ubuntu 10.04: Launch the following command (on a single line) to obtain all pre-requisite software packages.

```
sudo apt-get -y install libfontconfig1-dev libxrender-dev
 libpulse-dev swig g++ automake autoconf libtool Python-dev
 libfftw3-dev libcppunit-dev libboost-all-dev libusb-dev
 sdcc sdcc-libraries libsdl1.2-dev Python-wxgtk2.8 git-core
 guile-1.8-dev libqt4-dev Python-numpy ccache Python-opengl
 libgsl0-dev Python-cheetah Python-lxml doxygen qt4-dev-tools
 libqwt5-qt4-dev libqwtplot3d-qt4-dev pyqt4-dev-tools
 Python-qwt5-qt4 fort77
```

3.2 Installation

The general installation procedure is to run the following commands in order:

1. ./bootstrap
   (a) ./configure
   (b) make
   (c) make check
   (d) make install

Step 4 may generate errors, which depending on their particular type, may not be consequential. Searching the GNU Radio e-mail archives using an Internet search is recommended at this point. One common problem after installation may be that the appropriate path is not set for Python; in order to rectify this, append the following line to ./bashrc

```
export PYTHONPATH=insert path to gnuradio package directory
```

For Ubuntu 10.04: Download the .tar.gz release of GNU Radio from (*5*) and extract to a directory, e.g. "gnuradio". Launch the following commands:

```
cd gnuradio
./configure
make
make check
sudo make install
```

The "make check" command performs some basic checks to ensure the software built correctly. To test an actual GNU Radio application, launch the command

```
/usr/local/share/gnuradio/examples/audio/dial_tone.py
```

which launches a Python application that uses GNU Radio to generate two sinusoids and send their sum to the speakers, producing the sound of a dial tone.

3.3 USRP Configuration

Some Linux distributions do not provide default non-root access to handle hotplug USB devices. In this case, some care is needed to give the desired user access to the USRP. Distribution specific instructions are available on the GNU Radio Web site. In summary, one needs to proceed as follows:

1. Create a new group, call it "usrp".
2. Add the desired user to group usrp.
3. Create a new USB rule that tells Linux what to do when the USRP is plugged into a USB port of the machine.
4. Restart the machine.

For Ubuntu 10.04: Launch the following commands:

```
sudo addgroup usrp
sudo usermod -G usrp -a <YOUR_USERNAME>
echo 'ACTION==``add", BUS==``usb", SYSFS{idVendor}==``fffe",
 SYSFS{idProduct}==`0002", GROUP:=``usrp", MODE:=``0660"' tmpfile
sudo chown root.root tmpfile
sudo mv tmpfile /etc/udev/rules.d/10-usrp.rules
```

Next, restart the machine. To ensure the USRP is recognized, examine /dev/bus/usb after plugging in a USRP. Issuing the command:

```
ls -lR /dev/bus/usb | grep usrp
```

should give an output like:

```
crw-rw---- 1 root usrp 189, 514 Sep 01 09:46 003
```

Note well that it has been our experience that sometimes the USRP is not "fully" recognized after the first insertion into the USB or the first power-up after having been shut down. Examination of system logs shows that while Linux detects the presence of the USRP at the kernel level, it does not get configured properly. Applications attempting to

use the USRP in this state will generate errors that suggest the USRP is not connected at all. We have found that the simplest fix to this problem is simply to disconnect and reconnect the USRP.

3.4 Verifying the Installation

As mentioned earlier, the GNU Radio documentation recommends running "dial_tone.py" to verify proper GNU Radio installation (USRP need not be present). If all goes well, one should hear a dial tone at the speakers and have a good degree of confidence that GNU Radio has been correctly installed. We have found that this test does not always succeed, due to some possible quirks of the script perceiving a contention with the OS for sound card access. If at this step an error relating to the sound card driver is incurred and your sound-card works, then sometimes rerunning the command will solve the matter. If not, skip the audio test and proceed further to test with the USRP directly.

We would also like to verify that GNU Radio does in fact work with the USRP. GNU Radio provides a test script called "usrp_benchmark_usb", which helps in this regard. If GNU Radio can talk to the USRP, then at the least, this script should run properly. Otherwise, GNU Radio did not build properly or the USRP is not visible to the OS.

In the usrp/host/apps subdirectory of the GNU Radio home folder, launch the commands:

```
./test_usrp_standard_tx
./test_usrp_standard_rx
```

to test both the functionality of GNU Radio and USRP, and also obtain a good estimate of the maximum throughput between the host computer and the USRP. The maximum throughput is 32 MB/s.

In our experience, there are certain build configurations that may result in throughput of less than 32 MB/s. If this is the case, proceed with caution! Application performance may be very poor in this situation. Ensure that stable builds of the distribution and kernel are being used, with latest patches for the USB drivers. If operating within a VM like VMWare, ensure that USB 2.0 support is enabled. Note especially that other virtualization platforms, like VirtualBox, may have very poor throughputs due to poor implementations of virtualization for USB.

3.5 Troubleshooting

If the system cannot find the GNU Radio software base after installation, it might be that installation was done in a non-standard directory. In such a case, headers, libraries, and Python scripts are obscured from the application's view. To fix this, open the .bashrc file and append the following:

```
export PATH=$PATH:/opt/gnuradio/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/gnuradio/lib
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/opt/gnuradio/lib/pkgconfig
export PYTHONPATH=$PYTHONPATH:/opt/gnuradio/lib/Python2.6/site-packages
```

Log out and back in for the changes to take effect.

Some Linux distributions have packages that need to be downgraded in order to be compatible with GNU Radio. For example, the latest versions of gcc and Python can cause issues. If errors are encountered during the build process, try downgrading these.

If the error is related to a Python program, read the errors from the bottom up; the highest level trigger of the error will be found here. If using the GNU Radio built-in applications, searching the online archives for this error will be most helpful.

If the system is not recognizing the USRP, check the kernel message buffer via command "dmesg" to understand if the reason has to do with permissions or USB configuration.

If the ./test_usrp* scripts work fine, verifying host to USRP communication, but test applications verifying USRP to USRP traffic (e.g., benchmark_rx and benchmark_tx in /gnuradio-examples/Python/digital to test packet transmissions between two USRPs) do not work, then frequency offset issues are a prime suspect. Most GNU Radio test applications have a frequency offset correction flag. In order to discover the actual frequency offset between a pair of USRPs, a coarse but useful estimate can be obtained by generating a pure tone on the transmitting USRP at a given frequency, doing a fast Fourier transform (FFT) on the receiver at that same center frequency, and observing the extent of the shifted peak. In our experience, some pairs of USRPs will work with the GNU Radio test applications with no further action, but other pairs will have significant relative frequency offsets, which will cause GNU Radio applications to fail completely unless the offset is corrected for.

4. Understanding USRP Behavior

There are two important aspects of USRP behavior that are useful to know. The first relates to USRP behavior upon launching a GNU Radio application. When the USRP is first plugged in, the light-emitting diode (LED) to the right of the USB controller blinks at a fast rate of about 3 times per second, but slows down to about once per second when a GNU Radio application is using the USRP. Also, there is an audible change in the sound of the USRP cooling fan when a GNU Radio application is executing on the USRP. We have found that sometimes, applications for GNU Radio appear not to execute after being launched, but give no error messages. The issue comes up, for example, when launching one application soon after another and seems to have to do with the second application's inability to fully secure access to the USRP. In such a case, the obvious clue that the culprit is the USRP (and not the application) is the LED and fan behaving as if the USRP is in an idle state. In such a situation, terminate the application, wait a few moments, and restart it; else, restart the USRP itself.

The second aspect of the USRP that is important to be familiar with is the output of "O", "U", "u", or "a" characters when running a gnuradio program. These are codes indicating USRP status, with the following meaning:

- "u" = USRP
- "a" = audio (sound card)
- "O" = overrun (PC not keeping up with received data from the source, e.g., usrp)
- "U" = underrun (PC not providing data quickly enough to the sink, e.g., usrp)

Combining the flags results in a specific indication of PC to USRP communication status, for example, the following:

- "aUaU" indicates an audio underrun (the PC is not providing samples rapidly enough to the sound card).
- "uOuO" indicates that the PC is not keeping up with USRP sample rate, so USRP samples are being dropped.

As of the time of this writing, the frequency with which the symbols (e.g., "uO") are printed to the screen is *not* an indication of the frequency of the underlying error; overrun or underrun detection is implemented by polling, so the severity of the problem cannot be

determined by the frequency with which the above symbols alone are printed to the screen. This is slated to be corrected in a future release.

5. Conclusion

We have presented a high-level overview of the process of enabling a SDR testbed, covering the key steps involved in installation and configuration of the USRPs and GNU Radio. We have focused on those aspects that have received minimal attention in the official documentation for GNU Radio.

6. References

1. GNU Radio Web site. http://gnuradio.org/redmine/wiki/gnuradio (accessed 2011).

2. Ettus Research LLC Web site. http://www.ettus.com/products (accessed 2011).

3. GNU Radio Web site. Road map. http://gnuradio.org/redmine/projects/roadmap/gnuradio (accessed 2011).

4. GNU Radio Web site. Build Guide. http://gnuradio.org/redmine/wiki/gnuradio/BuildGuide (accessed 2011).

5. GNY Radio FTP site. ftp://ftp.gnu.org/gnu/gnuradio (accessed 2011).

NO. OF COPIES	ORGANIZATION
1 ELEC	ADMNSTR DEFNS TECHL INFO CTR ATTN DTIC OCP 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	US ARMY RSRCH DEV AND ENGRG CMND ARMAMENT RSRCH DEV & ENGRG CTR ARMAMENT ENGRG & TECHNLGY CTR ATTN AMSRD AAR AEF T J MATTS BLDG 305 ABERDEEN PROVING GROUND MD 21005-5001
1	US ARMY INFO SYS ENGRG CMND ATTN AMSEL IE TD A RIVERA FT HUACHUCA AZ 85613-5300
1	COMMANDER US ARMY RDECOM ATTN AMSRD AMR W C MCCORKLE 5400 FOWLER RD REDSTONE ARSENAL AL 35898-5000
1	US GOVERNMENT PRINT OFF DEPOSITORY RECEIVING SECTION ATTN MAIL STOP IDAD J TATE 732 NORTH CAPITOL ST NW WASHINGTON DC 20402
8	US ARMY RSRCH LAB ATTN IMNE ALC HRR MAIL & RECORDS MGMT ATTN RDRL CI J PELLEGRINO ATTN RDRL CIN A KOTT ATTN RDRL CIN T B RIVERA ATTN RDRL CIN T G VERMA ATTN RDRL CIN T P YU ATTN RDRL CIO LL TECHL LIB ATTN RDRL CIO MT TECHL PUB ADELPHI MD 20783-1197

INTENTIONALLY LEFT BLANK.